The Pragmatics of STAIRS

Ragnhild Kobro Runde\textsuperscript{1}, Øystein Haugen\textsuperscript{1}, and Ketil Stølen\textsuperscript{1,2}
\textsuperscript{1} Department of Informatics, University of Oslo, Norway
\textsuperscript{2} SINTEF ICT, Norway

Abstract. STAIRS is a method for the compositional development of interactions in the setting of UML 2.0. In addition to defining denotational trace semantics for the main aspects of interactions, STAIRS focuses on how interactions may be developed through successive refinement steps. In this tutorial paper, we concentrate on explaining the practical relevance of STAIRS. Guidelines are given on how to create interactions using the different STAIRS operators, and how these may be refined. The pragmatics is illustrated by a running example.

1 Introduction

STAIRS \cite{HHRS05a} is a method for the compositional development of interactions in the setting of UML 2.0 \cite{OMG05}. In contrast to e.g. UML state machines and Java programs, interactions are usually incomplete specifications, typically describing example runs of the system. STAIRS is designed to deal with this incompleteness. Another important feature of STAIRS is the possibility to distinguish between alternative behaviours representing underspecification and alternative behaviours that must all be present in an implementation, for instance due to inherent nondeterminism.

STAIRS is not intended to be a complete methodology for system development, but should rather be seen as a supplement to methodologies like e.g. RUP \cite{Kru04}. In particular, STAIRS focuses on refinement, which is a development step where the specification is made more complete by information being added to it in such a way that any valid implementation of the refined specification will also be a valid implementation of the original specification.

In this paper we focus on refinement relations. We define general refinement, which in turn has four special cases referred to as narrowing, supplementing, detailing and limited refinement. Narrowing means to reduce the set of possible system behaviours, thus reducing underspecification. Supplementing, on the other hand, means to add new behaviours to the specification, taking into account the incomplete nature of interactions, while detailing means to add more details to the specification by decomposing lifelines. By general refinement, the nondeterminism required of an implementation may be increased freely, while limited refinement is a special case restricting this possibility.

Previous work on STAIRS has focused on its basic ideas, explaining the various concepts such as the distinction between underspecification and inherent nondeterminism \cite{HHRS05a,RHS05b}, time \cite{HHRS05b}, and negation \cite{RHS05a}, as
well as how these are formalized. In this paper, we take the theory of STAIRS one step further, focusing on its practical consequences by giving practical guidelines on how to use STAIRS. In particular, we explain how to use the various STAIRS operators when creating specifications in the form of interactions, and how these specifications may be further developed through valid refinement steps.

The paper is organized as follows: In Sect. 2 we give a brief introduction to interactions and their semantic model as we have defined it in STAIRS. Section 3 is an example-guided walkthrough of the main STAIRS operators for creating interactions, particularly focusing on alternatives and negation. For each operator we give both its formal definition and guidelines for its practical usage. Section 4 gives the pragmatics of refining interactions. In Sect. 5 we explain how STAIRS relates to other similar approaches, in particular UML 2.0, while we give some concluding remarks in Sect. 6.

2 The Semantic Model of STAIRS

In this section we give a brief introduction to interactions and their trace semantics as defined in STAIRS. The focus here is on the semantic model. Definitions of concrete syntactical operators will mainly be presented together with the discussion of these operators later in this paper. For a thorough account of the STAIRS semantics, see [HHRS05b] and the extension with data in [RHS05b].

An interaction describes one or more positive (i.e. valid) and/or negative (i.e. invalid) behaviours. As a very simple example, the sequence diagram in Fig. 1 specifies a scenario in which a client sends the message cancel(appointment) to an appointment system, which subsequently sends the message appointment-Cancelled back to the client, together with a suggestion for a new appointment to which the client answers with the message yes. The client finally receives the message appointmentMade.

Formally, we use denotational trace semantics to explain the meaning of a single interaction. A trace is a sequence of events, representing a system run. The most typical examples of events are the sending and the reception of a message, where a message is a triple (s, tr, re) consisting of a signal s, a transmitter lifeline tr and a receiver lifeline re. For a message m, we let !m and ?m denote the sending and the reception of m, respectively. As will be explained in Sect. 3.2, we also have some special events representing the use of data in e.g. constraints and guards.

The diagram in Fig. 1 includes ten events, two for each message. These are combined by the implicit weak sequencing operator seq, which will be formally defined at the end of this section. Informally, the set of traces described by such a diagram is the set of all possible sequences consisting of its events such that the send event is ordered before the corresponding receive event, and events on the same lifeline are ordered from top down. Shortening each message to the first and the capitalized letter of its signal, we thus get that Fig. 1 specifies two positive traces (\langle !c, ?c, !AC, ?aC, !aS, ?aS, !y, ?y, !aM, ?aM \rangle) and (\langle !c, ?c, !aC, !aS, ?aC, ?aS, !y, ?y, !aM, ?aM \rangle), where the only difference is the